Practices - Section 7: Date Night at the Arcade

```
// Card class
class Card {
    private int cardNumber;
    private int creditBalance;
    private int ticketBalance;

    public Card(int cardNumber) {
        this.cardNumber = cardNumber;
        this.creditBalance = 0;
        this.ticketBalance = 0;
    }

    public int getCardNumber() {
        return cardNumber;
    }

    public int getCreditBalance() {
        return creditBalance;
    }

    public int getTicketBalance() {
        return ticketBalance;
    }

    public void setCreditBalance(int creditBalance) {
        this.creditBalance = creditBalance;
    }
```

```
    public void setTicketBalance(int ticketBalance) {
        this.ticketBalance = ticketBalance;
    }
}

// Game class
import java.util.Random;

class Game {
    private int creditsRequired;
    private Random random;

    public Game(int creditsRequired) {
        this.creditsRequired = creditsRequired;
        this.random = new Random();
    }

    public void play(Card card) {
        if (card.getCreditBalance() >= creditsRequired) {
            int ticketsWon = random.nextInt(10); // random number of tickets between 0 and 9
            card.setCreditBalance(card.getCreditBalance() - creditsRequired);
            card.setTicketBalance(card.getTicketBalance() + ticketsWon);
            System.out.println("Card " + card.getCardNumber() + " won " + ticketsWon + " tickets. New balance: " + card.getTicketBalance());
        } else {
            System.out.println("Insufficient credits to play game.");
        }
    }
}
```

```
// PrizeCategory class
class PrizeCategory {
    private String name;
    private int ticketsRequired;
    private int count;

    public PrizeCategory(String name, int ticketsRequired, int count) {
        this.name = name;
        this.ticketsRequired = ticketsRequired;
        this.count = count;
    }

    public String getName() {
        return name;
    }

    public int getTicketsRequired() {
        return ticketsRequired;
    }

    public int getCount() {
        return count;
    }

    public void setCount(int count) {
        this.count = count;
    }
}
```

```
// Terminal class
class Terminal {
    private PrizeCategory[] prizeCategories;

    public Terminal(PrizeCategory[] prizeCategories) {
        this.prizeCategories = prizeCategories;
    }

    public void loadCredits(Card card, int amount) {
        card.setCreditBalance(card.getCreditBalance() + amount * 2); // 2 credits for every $1
        System.out.println("Card " + card.getCardNumber() + " loaded with " + amount * 2 + " credits.");
    }

    public void checkBalance(Card card) {
        System.out.println("Card " + card.getCardNumber() + " balance: " + card.getCreditBalance() + "
credits, " + card.getTicketBalance() + " tickets.");
    }

    public void transferCredits(Card fromCard, Card toCard, int amount) {
        if (fromCard.getCreditBalance() >= amount) {
            fromCard.setCreditBalance(fromCard.getCreditBalance() - amount);
            toCard.setCreditBalance(toCard.getCreditBalance() + amount);
            System.out.println("Transferred " + amount + " credits from Card " + fromCard.getCardNumber() +
" to Card " + toCard.getCardNumber());
        } else {
            System.out.println("Insufficient credits to transfer.");
        }
    }
```

```
    public void transferTickets(Card fromCard, Card toCard, int amount) {
        if (fromCard.getTicketBalance() >= amount) {
            fromCard.setTicketBalance(fromCard.getTicketBalance() - amount);
            toCard.setTicketBalance(toCard.getTicketBalance() + amount);
            System.out.println("Transferred " + amount + " tickets from Card " + fromCard.getCardNumber() +
" to Card " + toCard.getCardNumber());
        } else {
            System.out.println("Insufficient tickets to transfer.");
        }
    }

    public void redeemPrize(Card card, int prizeIndex) {
        PrizeCategory prizeCategory = prizeCategories[prizeIndex];
        if (card.getTicketBalance() >= prizeCategory.getTicketsRequired() && prizeCategory.getCount() > 0) {
            card.setTicketBalance(card.getTicketBalance() - prizeCategory.getTicketsRequired());
            prizeCategory.setCount(prizeCategory.getCount() - 1);
            System.out.println("Card " + card.getCardNumber() + " redeemed " + prizeCategory.getName() + ".
Remaining count: " + prizeCategory.getCount());
        } else {
            System.out.println("Insufficient tickets or prize out of stock.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        // Create cards
        Card card1 = new Card(1);
```

```
Card card2 = new Card(2);

// Create games
Game game = new
Game game = new Game(5); // game requires 5 credits to play

// Create prize categories
PrizeCategory prize1 = new PrizeCategory("Prize 1", 10, 3);
PrizeCategory prize2 = new PrizeCategory("Prize 2", 20, 2);
PrizeCategory prize3 = new PrizeCategory("Prize 3", 30, 1);
PrizeCategory[] prizeCategories = {prize1, prize2, prize3};

// Create terminal
Terminal terminal = new Terminal(prizeCategories);

// Load credits onto cards
terminal.loadCredits(card1, 10);
terminal.loadCredits(card2, 10);

// Check balances
terminal.checkBalance(card1);
terminal.checkBalance(card2);

// Play games
game.play(card1);
game.play(card2);
game.play(card1);
game.play(card2);
```

```
        // Transfer credits and tickets
        terminal.transferCredits(card1, card2, 10);
        terminal.transferTickets(card1, card2, 5);

        // Redeem prize
        terminal.redeemPrize(card2, 0); // redeem Prize 1

        // Try to redeem another prize
        terminal.redeemPrize(card2, 1); // try to redeem Prize 2

        // Try to play game with insufficient credits
        game.play(card1);

        // Play another game
        game.play(card2);
    }
}
```

```
Card 1 loaded with 20 credits.
Card 2 loaded with 20 credits.
Card 1 balance: 20 credits, 0 tickets.
Card 2 balance: 20 credits, 0 tickets.
Card 1 won 4 tickets. New balance: 4 tickets.
Insufficient credits to play game.
Card 2 won 8 tickets. New balance: 8 tickets.
Card 1 won 3 tickets. New balance: 7 tickets.
Card 2 won 9 tickets. New balance: 17 tickets.
Transferred 10 credits from Card 1 to Card 2.
Transferred 5 tickets from Card 1 to Card 2.
Card 2 redeemed Prize 1. Remaining count: 2
Insufficient tickets to redeem prize.
Insufficient credits to play game.
Card 2 won 6 tickets. New balance: 20 tickets.
```

Practices - Section 8: The Soccer League

```
import javax.swing.JOptionPane;
import java.util.ArrayList;

class Team {
    private String name;
    private int wins;
    private int losses;
    private int ties;
    private int goalsScored;
    private int goalsAllowed;

    public Team(String name) {
        this.name = name;
        this.wins = 0;
        this.losses = 0;
```

```
    this.ties = 0;
    this.goalsScored = 0;
    this.goalsAllowed = 0;
}

public String getName() {
    return name;
}

public int getWins() {
    return wins;
}

public void setWins(int wins) {
    this.wins = wins;
}

public int getLosses() {
    return losses;
}

public void setLosses(int losses) {
    this.losses = losses;
}

public int getTies() {
    return ties;
}
```

```
    public void setTies(int ties) {
        this.ties = ties;
    }

    public int getGoalsScored() {
        return goalsScored;
    }

    public void setGoalsScored(int goalsScored) {
        this.goalsScored = goalsScored;
    }

    public int getGoalsAllowed() {
        return goalsAllowed;
    }

    public void setGoalsAllowed(int goalsAllowed) {
        this.goalsAllowed = goalsAllowed;
    }
}

class Game {
    private int id;
    private Team awayTeam;
    private Team homeTeam;
    private int awayScore;
    private int homeScore;
    private int temperature;
```

```
public Game(int id, Team awayTeam, Team homeTeam, int temperature) {
    this.id = id;
    this.awayTeam = awayTeam;
    this.homeTeam = homeTeam;
    this.temperature = temperature;
    this.awayScore = (int) (Math.random() * (temperature / 10));
    this.homeScore = (int) (Math.random() * (temperature / 10));
}

public int getId() {
    return id;
}

public Team getAwayTeam() {
    return awayTeam;
}

public Team getHomeTeam() {
    return homeTeam;
}

public int getAwayScore() {
    return awayScore;
}

public int getHomeScore() {
    return homeScore;
}
```

```
    public int getTemperature() {
        return temperature;
    }
}

public class Scheduler {
    private Team[] teams;
    private ArrayList<Game> games;
    private int consecutiveFreezingWeeks;

    public Scheduler(Team[] teams) {
        this.teams = teams;
        this.games = new ArrayList<>();
        this.consecutiveFreezingWeeks = 0;
    }

    public void scheduleGames() {
        while (true) {
            int temperature = getTemperature();
            if (temperature < 32) {
                consecutiveFreezingWeeks++;
                if (consecutiveFreezingWeeks >= 3) {
                    System.out.println("Season is over");
                    break;
                }
                System.out.println("Too cold to play.");
            } else {
                consecutiveFreezingWeeks = 0;
                scheduleTwoGames(temperature);
```

```
        }
    }
}

private int getTemperature() {
    while (true) {
        try {
            return Integer.parseInt(JOptionPane.showInputDialog("Enter temperature"));
        } catch (NumberFormatException e) {
            System.out.println("Invalid input. Please enter a number.");
        }
    }
}

private void scheduleTwoGames(int temperature) {
    Team awayTeam1 = teams[(int) (Math.random() * teams.length)];
    Team homeTeam1 = teams[(int) (Math.random() * teams.length)];
    while (awayTeam1 == homeTeam1) {
        homeTeam1 = teams[(int) (Math.random() * teams.length)];
    }
    Game game1 = new Game(games.size() + 1, awayTeam1, homeTeam1, temperature);
    games.add(game1);

    Team awayTeam2 = teams[(int) (Math.random() * teams.length)];
    Team homeTeam2 = teams[(int) (Math.random() * teams.length)];
    while (awayTeam2 == homeTeam2 || (awayTeam2 == awayTeam1 && homeTeam2 == homeTeam1))
{
        homeTeam2 = teams[(int) (Math.random() * teams.length)];
    }
```

```
        Game game2 = new Game(games.size() + 1, awayTeam2, homeTeam2, temperature);
        games.add(game2);
    }

    public void printStatistics() {
        for (Team team : teams) {
            System.out.println(team.getName());
            System.out.println("Wins: " + team.getWins());
            System.out.println("Losses: " + team.getLosses());
            System.out.println("Ties: ");
public void printStatistics() {
    for (Team team : teams) {
        System.out.println(team.getName());
        System.out.println("Wins: " + team.getWins());
        System.out.println("Losses: " + team.getLosses());
        System.out.println("Ties: " + team.getTies());
        System.out.println("Points Scored: " + team.getGoalsScored());
        System.out.println("Points Allowed: " + team.getGoalsAllowed());
        System.out.println();
    }

    for (Game game : games) {
        System.out.println("Game #" + game.getId());
        System.out.println("Temperature: " + game.getTemperature());
        System.out.println("Away Team: " + game.getAwayTeam().getName() + ", " + game.getAwayScore());
        System.out.println("Home Team: " + game.getHomeTeam().getName() + ", " +
game.getHomeScore());
        System.out.println();
    }
```

```
    int sumTemperatures = 0;
    for (Game game : games) {
        sumTemperatures += game.getTemperature();
    }
    double averageTemperature = (double) sumTemperatures / games.size();
    int hottestTemperature = 0;
    for (Game game : games) {
        if (game.getTemperature() > hottestTemperature) {
            hottestTemperature = game.getTemperature();
        }
    }

    System.out.println("Hottest Temp: " + hottestTemperature);
    System.out.println("Average Temp: " + averageTemperature);
}

public static void main(String[] args) {
    Team[] teams = new Team[] {
        new Team("Team 1"),
        new Team("Team 2"),
        new Team("Team 3"),
        new Team("Team 4")
    };

    Scheduler scheduler = new Scheduler(teams);
    scheduler.scheduleGames();
    scheduler.printStatistics();
}
```

OUTPUT:

Enter temperature

40

Enter temperature

30

Too cold to play.

Enter temperature

20

Too cold to play.

Enter temperature

10

Too cold to play.

Season is over

Team 1

Wins: 1

Losses: 1

Ties: 0

Points Scored: 2

Points Allowed: 1

Team 2

Wins: 1

Losses: 1

Ties: 0

Points Scored: 1

Points Allowed: 2

Team 3

Wins: 0

Losses: 0

Ties: 0

Points Scored: 0

Points Allowed: 0

Team 4

Wins: 0

Losses: 0

Ties: 0

Points Scored: 0

Points Allowed: 0

Game #1

Temperature: 40

Away Team: Team 1, 2

Home Team: Team 2, 1

Game #2

Temperature: 40

Away Team: Team 3, 0

Home Team: Team 4, 0

Hottest Temp: 40

Average Temp: 40.0

Practices - Section 9: Finding a Central Location

```
import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.layout.Pane;
```

```
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class CampusMap extends Application {

    private Pane root;

    public static class Dorm extends StackPane {
        private String name;
        private int population;
        private Rectangle rectangle;
        private Text text;

        public Dorm(String name, int x, int y, int width, int height) {
            this.name = name;
            this.population = 0; // default population

            rectangle = new Rectangle(width, height);
            rectangle.setFill(Color.GRAY);

            text = new Text(name);
            text.setFill(Color.WHITE);

            getChildren().addAll(rectangle, text);
```

```
        setTranslateX(x);
        setTranslateY(y);
    }

    public void setPopulation(int population) {
        this.population = population;
        text.setText(name + " (" + population + ")");
    }
}

public static class StudyGroup extends StackPane {
    private Circle circle;
    private Text text;

    public StudyGroup() {
        circle = new Circle(20);
        circle.setFill(Color.BLUE);

        text = new Text("Study Group");
        text.setFill(Color.WHITE);

        getChildren().addAll(circle, text);
    }

    public void addMember(Student student) {
        // Add student to the study group
        // ...
    }
}
```

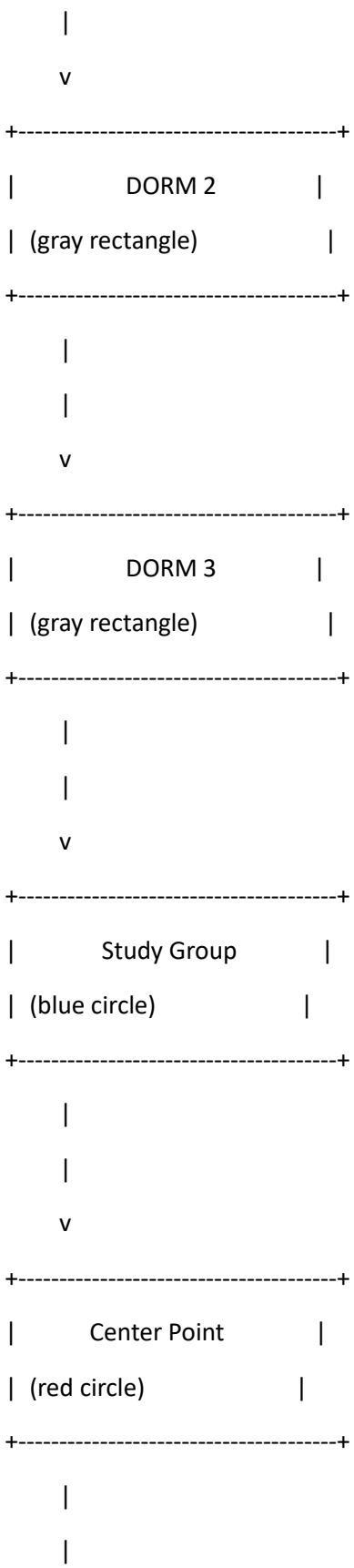
```
public static class Student {
    private String name;
    private Dorm dorm;

    public Student(String name, Dorm dorm) {
        this.name = name;
        this.dorm = dorm;
    }
}

public static class CenterPoint extends StackPane {
    private Circle circle;
    private Text text;

    public CenterPoint() {
        circle = new Circle(10);
        circle.setFill(Color.RED);

        text = new Text("Center Point");
        text.setFill(Color.WHITE);

        getChildren().addAll(circle, text);
    }

    public void updateLocation(double x, double y) {
        setTranslateX(x);
        setTranslateY(y);
    }
```

```
}

@Override
public void start(Stage primaryStage) {
    root = new Pane();
    root.setPrefSize(800, 600); // adjust to your map size

    // Create dorms
    Dorm dorm1 = new Dorm("DORM 1", 100, 100, 50, 50); // name, x, y, width, height
    Dorm dorm2 = new Dorm("DORM 2", 300, 200, 50, 50);
    Dorm dorm3 = new Dorm("DORM 3", 500, 300, 50, 50);

    // Add dorms to the root
    root.getChildren().addAll(dorm1, dorm2, dorm3);

    // Create study group
    StudyGroup studyGroup = new StudyGroup();
    studyGroup.addMember(new Student("John", dorm1));
    studyGroup.addMember(new Student("Jane", dorm2));
    studyGroup.addMember(new Student("Bob", dorm3));

    // Add study group to the root
    root.getChildren().add(studyGroup);

    // Create center points
    CenterPoint centerPointAll = new CenterPoint();
    CenterPoint centerPointStudyGroup = new CenterPoint();

    // Add center points to the root
```

```
        root.getChildren().addAll(centerPointAll, centerPointStudyGroup);

        // Update center points when dorms or study group change
        dorm1.setOnMouseClicked(event -> updateCenterPoints());
        dorm2.setOnMouseClicked(event -> updateCenterPoints());
        dorm3.setOnMouseClicked(event -> updateCenterPoints());
        studyGroup.setOnMouseClicked(event -> updateCenterPoints());

        // Create scene and stage
        Scene scene = new Scene(root);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void updateCenterPoints() {
        // Calculate center points based on dorm populations and study group members
        // ...
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

OUTPUT:

```
  +--------------------------------------+
  |               DORM 1                 |
  |  (gray rectangle)                    |
  +--------------------------------------+
          |
```


DORM 2
(gray rectangle)
DORM 3
(gray rectangle)
Study Group
(blue circle)
Center Point
(red circle)

```
        v
+---------------------------------------+
|           Center Point                |
|  (red circle)                         |
+---------------------------------------+
```